Howto: C-Programmierung: Arrays und Zeiger

Wednesday, 7 September 2022 09:12

Alle Variablen, mit denen wir bis jetzt gearbeitet haben hat der Mikrocontroller im SRAM abgelegt

Wie ist der SRAM aufgebaut?

Jedes Kästchen steht für ein Byte aus dem SRAM

Genauer aus dem AVR-Datasheet Kapitel 7.3 / Seite 18

Der ATmega328p hat 2kB = 2048 Bytes SRAM

Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Registers	0x0060 - 0x00FF
Internal SRAM (~~1048~~ 2048 x 8)	0x0100 0x03FF 0x08FF

Beachte die Fehler in dem Datenblatt?

Jedes Byte im SRAM des Mikrocontrollers hat eine eigene Adresse. Die Adressen im ATmega328p sind 16-bit groß.

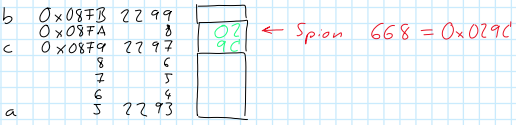
Mit dem & - Operator kann man sich die Adresse einer Variablen beschaffen.

Ein Zeiger (eng pointer) ist nichts anderes als die Adresse für einen bestimmten Datentyp

→ Legen Sie nacheinander folgende Variablen an:

1) long
2) byte
3) int

Lassen Sie sich die Adressen der 3 Variablen ausgeben.




```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  long a; //4 bytes
  byte b; //1 byte
  int  c = 668; //2 bytes 668 = 0x029C
  // byte spion;
  // &spion = 0x08FC;
  byte *spion;
  spion = (byte *) &c + 1;
  // spion = (byte *) 0x08F9;
  //nur zum nachschauen von der Adressse von a: &a

  Serial.print("a: ");
  Serial.println(a);
  Serial.println((int) &a);
  Serial.print("b: ");
  Serial.println(b);
  Serial.println((int) &b);
  Serial.print("c: ");
  Serial.println(c);
  Serial.println((int) &c);
  Serial.print("Spion: ");
  Serial.println(*spion, HEX);
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

Wieso sind die Adressen von Variablen überhaupt wichtig?

① Bei jedem Funktionsaufruf müssen die Werte der Argumente kopiert werden "Call by value"
Für "große" Argumente (Structures, Arrays) ist das natürlich sehr zeitaufwendig.
→ Ausweg: Der Funktion wird nur eine Adresse des Argumentes (Structure, Array) übergeben.
es werden also nur 16-Bit kopiert (Call by reference). Das hat aber sofort zur Folge,
dass die Funktion nun (dorch die Adresse) mit den orginal Daten arbeitet. Also
jede Veränderung der referenzierten Variable ist sofort in der Hauptfunktion (loop, main)
sichtbar.
Warum? Weil wir mit dem Orginal arbeiten nicht mit einer Kopie.

② Arrays werden in C nur durch ihre Adresse repräsentiert
siehe unten

③ malloc und free (nicht in der Arduino Referenz)

④ Viele viele weitere zB
linked-Lists
function-pointer
...

Das Gegenstück zum &-Operator ist der *-Operator: * - Operator

Mit *p holt man den "Inhalt der Speicherzelle" mit der Adresse p.

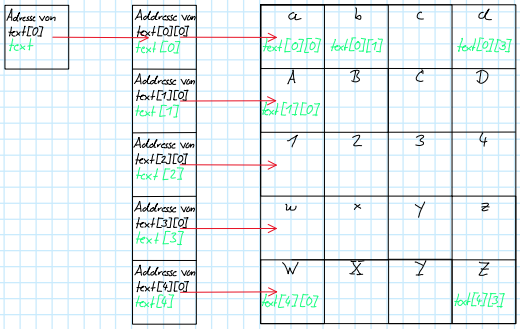
Wieviele Byks für den "Inhalt der Speicherzelle" benutzt werden geht aus dem Datentyp für *p hervor.

→ Erstelle eine "call by reference" Funktion, die einen Zahlenwert verdoppelt.

```
int zahl = 10;
void setup() {
  Serial.begin(9600);
  Serial.println("Vor der Verdopplung: " + String(zahl));
  *(&zahl) = *(&zahl) * 2;
  Serial.println("Nach der Verdopplung: " + String(zahl));
}
void loop() {
  // Leere Schleife
}
```

Inhalt der Zelle

Variable für die Zelle



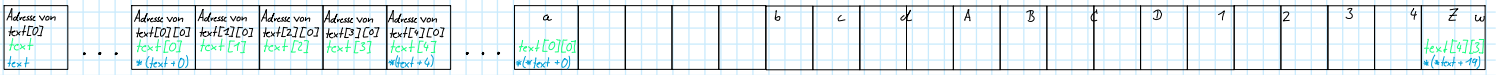
text[0][3] ist eine Abkürzung für *(text[0]+3)
und das ist eine Abkürzung für *(*(text+0)+3)
text[0][3] ≙ *(text[0]+3) ≙ *(*(text+0)+3)

&: liefere die Adresse einer Variablen

*: liefere den Inhalt einer Adresse

* text[4] ist 'W'

In Wirklichkeit ist die Anordnung im SRAM eher folgendermassen:



Adressen, Zeiger, Arrays, Call by Reference und Zeigerarithmetik

Deklaration

? Dimension von Index 0 3

char text[5][4];

Datentyp

1 Dimension von Index 0 4

Übungen

Was wäre ein Serial println jeweils ausgeben?

*text[0]

*(text[0]+2)

*(text[0]+5)

& text[0]